BlackBerry Java Application MIDlet

Versión: 5.0

Guía de desarrollo

BlackBerry

Publicado: 2010-04-21
SWD-810881-0421092922-005

Contenido

1 API de Java ME y Java para BlackBerry 3

2 Ventajas de los MIDlets 4

3 Desventajas de los MIDlets 5

4 Creación de un MIDlet básico 6

5 Mejora de un MIDlet básico 7

Mostrar elementos en la pantalla 7

Mostrar una imagen 7

Ejemplo de código: mostrar una imagen 9

Dibujar en la pantalla 10

Dibujar un rectángulo 11

Ejemplo de código: dibujar un rectángulo 12

Dibujar un círculo 12

Ejemplo de código: dibujar un círculo 14

Mostrar los controles 14

Mostrar una etiqueta 15

Ejemplo de código: mostrar un ejemplo 17

Mostrar un campo de texto 17

Ejemplo de código: mostrar un campo de texto 19

Mostrar botones de opción 19

Ejemplo de código: mostrar botones de opción 22

Mostrar casillas de verificación 22

Ejemplo de código: mostrar casillas de verificación 24

Comandos para la pantalla 25

Ejemplo de código: mostrar comandos 27

Mostar un menú desplegable 27

Ejemplo de código: mostrar un menú desplegable 29

Gestionar la entrada 30

Responder a comandos 30

Ejemplo de código: responder a comandos 32

Responder a la entrada de controles 33

6 Glosario 54

7 Comentarios 55

8 Historial de revisiones del documento 56

9 Aviso legal 57

API de Java ME y Java para BlackBerry 1

Java® ME es una plataforma estándar del sector que define los conjuntos comunes de API de Java para distintos tipos de dispositivos inalámbricos e incorporados. Una aplicación Java ME en un dispositivo BlackBerry® se ejecuta en la máquina virtual BlackBerry® Java® Virtual Machine, que proporciona todos los servicios de tiempo de ejecución a las aplicaciones y realiza funciones tales como asignaciones normales de memoria, comprobaciones de seguridad y recogida de datos desechables.

La MIDP estándar de Java ME gestiona las necesidades de la API y BlackBerry JVM de un dispositivo inalámbrico restringido con una interfaz de usuario. El dispositivo BlackBerry es compatible con la MIDP estándar de Java ME tal como se define en JSR 118. BlackBerry que ejecutan BlackBerry Device Software versión 5.0 o posterior son compatibles con MIDP 2.1. La MIDP estándar de Java ME proporciona un conjunto fundamental de API de Java que son compatibles con cualquier dispositivo BlackBerry, independientemente de su sistema operativo subyacente. Los desarrolladores a menudo pueden construir una aplicación Java utilizando la API MIDP estándar y ejecutando esa aplicación en muchos tipos diferentes de dispositivos.

Ventajas de los MIDlets

2

Aunque las API de BlackBerry® Java® le proporcionan acceso a más características que la API de MIDP, los MIDlets presentan algunas ventajas sobre las aplicaciones BlackBerry Java Applications.

Ventaja	Descripción
Plataforma cruzada	Los MIDlets se ejecutan en cualquier dispositivo basado en Java que sea compatible con la versión de CLDC y MIDP que utiliza el MIDlet.
Disponibilidad de los recursos de aprendizaje	Puesto que los MIDlets se pueden ejecutar en varios dispositivos, muchos proveedores publican información acerca del desarrollo de MIDlets. Muchos ejemplos de código de MIDlets están disponibles al público, así como los libros, artículos y sitios Web acerca del desarrollo de MIDlets.
API de interfaz de usuario con subprocesos seguros	Excepto el método `ServiceRepaints()` de la clase `Canvas`, puede utilizar las API de la interfaz de usuario sin limitaciones originadas por los problemas de subprocesos.

Desventajas de los MIDlets

3

Las ventajas que presentan los MIDlets sobre las aplicaciones BlackBerry® Java® Applications también generan sus inconvenientes.

Desventaja	Descripción
Acceso limitado a la característica	Muchas de las características de los dispositivos BlackBerry están disponibles gracias a la API de BlackBerry Java. Sólo un subconjunto de dichas características podrá aprovecharse a través de la API de MIDlet.
Funcionalidad de diseño de pantalla de baja calidad	La API de MIDlet proporciona únicamente los mecanismos más sencillos con el fin de posicionar fácilmente los elementos de la interfaz de usuario en la pantalla del dispositivo. El resultado es que, incluso para los diseños relativamente más sencillos, es necesario recurrir a la escritura de código para dibujar directamente en la pantalla.

Creación de un MIDlet básico

4

Los MIDlets son aplicaciones inalámbricas que se ejecutan en un entorno de tiempo de ejecución muy controlado. El software de administración de la aplicación gestiona el ciclo de vida de un MIDlet, lo cual genera su transición entre tres estados: activo, en pausa y destruido. La clase `MIDlet` representa la aplicación y especifica la interfaz a implementar para activar el software de administración de aplicaciones para controlar el MIDlet.

Para crear el MIDlet más básico, debe extender la clase MIDlet y proporcionar implementaciones para los tres métodos abstractos de la clase: `startApp()`, `pauseApp()` y `destroyApp()`. La clase `MIDlet` se implementa en la biblioteca `java.microedition.midlet`, por lo que debe importar dicha biblioteca para crear un MIDlet.

La siguiente lista ilustra el código necesario para crear el MIDlet más básico.

```
import javax.microedition.midlet.*;
public class MostBasicMIDlet extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

Para conocer más detalles acerca de la clase `MIDlet`, consulte la referencia API de BlackBerry® Java® Development Environment.

Mejora de un MIDlet básico

5

Tras compilar e instalar correctamente un MIDlet básico, dispondrá de una base firme sobre la que construir. Para mejorar un MIDlet, agregue una pequeña cantidad de nuevas funcionalidades, por ejemplo, un nuevo control de interfaz de usuario. Una vez agregada la funcionalidad, compile y pruebe el MIDlet antes de mejorarlo en mayor medida posteriormente.

Mostrar elementos en la pantalla

Para mostrar elementos en la pantalla del dispositivo BlackBerry® puede utilizar las clases UI integradas o bien utilizar sus propias clases UI. Las clases UI integradas modelan las pantallas y los elementos que puede incluir en las pantallas. Puede crear rápidamente una clase UI mediante las clases UI integradas, aunque éstas no son muy flexibles. El otro enfoque consiste en crear sus propias clases UI.

Para crear sus propias clases UI, puede extender la clase `CustomItem` o dibujar directamente en la pantalla omitiendo el método `paint()` de la clase `Canvas`. El enfoque que utiliza la clase `Canvas` es muy flexible, pero requiere un esfuerzo considerable. Los dispositivos MIDP tienen tamaño de pantalla muy variables. Para dibujar elementos de la interfaz de usuario debe recuperar mediante programación la métrica de pantalla y utilizar la métrica para posicionar y medir adecuadamente los elementos de la interfaz de usuario.

Mostrar una imagen

Para mostrar una imagen en la pantalla de un dispositivo BlackBerry®, cree un objeto `Image` y rellénelo llamando al método estático `Image.createImage()`. Proporcione la ubicación de la imagen como parámetro.

Debe agregar la imagen al proyecto. En BlackBerry® Java® Development Environment, compruebe que se muestra junto con los archivos de origen de Java® en el control de árbol dentro de BlackBerry® Integrated Development Environment.

1. Importe las tres bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DisplayAnImage extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
```

```
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de las clases `Display` y `Form`. Debe crear también una variable privada en la cual almacenar la imagen.

```
public class DisplayAnImage extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private Image pngBackground;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ImageItem`, rellénela con una imagen y agréguela a un objeto `Form`.
   a. Utilice el método estático `createImage()` de la clase `Image` para crear una instancia de la clase `Image` que representa la imagen que desea mostrar.

```
public void startApp()
{
   pngBackground = Image.createImage("test_image.png");
}
```

   b. Cree una nueva instancia de la clase `ImageItem`. El constructor toma como parámetros una etiqueta, un objeto Image, un diseño, texto alternativo y un modo de apariencia.

```
public void startApp()
{
   pngBackground = Image.createImage("test_image.png");
   ImageItem img = new ImageItem("bg",pngBackground,
   ImageItem.LAYOUT_EXPAND,"background",ImageItem.PLAIN);
}
```

   c. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar la imagen al formulario.

```
public void startApp()
{
   pngBackground = Image.createImage("test_image.png");
   ImageItem img = new ImageItem("bg",pngBackground,
   ImageItem.LAYOUT_EXPAND,"background",ImageItem.PLAIN);
```

```
    mForm = new Form("MIDlet Developer Guide: Display an image.");
    mForm.append(img);
}
```

d. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
    pngBackground = Image.createImage("test_image.png");
    ImageItem img = new ImageItem("bg",pngBackground,
    ImageItem.LAYOUT_EXPAND,"background",ImageItem.PLAIN);
    mForm = new Form("MIDlet Developer Guide: Display an image.");
    mForm.append(img);
    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

e. Agregue un bloque try-catch en caso de no poder acceder al archivo de imagen.

```
public void startApp()
{
    try
    {
        pngBackground = Image.createImage("test_image.png");
        ImageItem img = new ImageItem("bg",pngBackground,
        ImageItem.LAYOUT_EXPAND,"background");
        mForm = new Form("MIDlet Developer Guide: Display an image.");
        mForm.append(img);
        mDisplay = Display.getDisplay(this);
        mDisplay.setCurrent(mForm);
    }
    catch(IOException e)
    {
        mForm.append(e.getMessage());
    }
}
```

Ejemplo de código: mostrar una imagen

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class DisplayAnImage extends MIDlet
{
    private Form mForm;
```

```
    private Display mDisplay;
    private Image pngBackground;
    public DisplayAnImage()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
    public void startApp()
    {
            mDisplay = Display.getDisplay(this);
            mForm = new Form("MIDlet Developer Guide: Display an image.");
            try
            {
                pngBackground = Image.createImage("test_image.png");
                ImageItem img = new ImageItem("bg",pngBackground,
                ImageItem.LAYOUT_EXPAND,"background");
                mForm.append(img);
                mDisplay.setCurrent(mForm);
            }
            catch(IOException e)
            {
                System.out.println(e.getMessage());
            }
    }
    public void pauseApp()
    {
    }
}
```

Dibujar en la pantalla

Para dibujar en la pantalla debe extender la clase `Canvas` y omitir el método `paint()`. Para volver a dibujar la pantalla, el entorno de aplicaciones MIDlet llama al método `paint()` y pasa al método un objeto `Graphics`. El objeto proporciona acceso a la información sobre la pantalla y le permite dibujar formas simples y presentar imágenes y texto.

Ejemplo de código: omitir el método paint() para dibujar un círculo coloreado

En el siguiente ejemplo de código, se utilizan los métodos `getWidth()` y `getHeight()` para recuperar las dimensiones de la pantalla en el dispositivo BlackBerry® que ejecuta el código. A continuación, el círculo se coloca cerca del centro de la pantalla especificando la mitad de la anchura y la mitad de la altura como los dos primeros valores de parámetro en `drawArc()` y `fillArc()`.

```
class ShapeCanvas extends Canvas
{
    public void paint(Graphics g)
    {
```

```
        int width  = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawArc(width/2,height/2,50,50,0,360);
        g.fillArc(width/2,height/2,50,50,0,360);
    }
}
```

Dibujar un rectángulo

Para dibujar un rectángulo en la pantalla del dispositivo BlackBerry®, debe extender la clase `Canvas` y omitir su método `paint()`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Para crear el marco para el MIDlet, extienda la clase `MIDlet` y omita los métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Extienda la clase `Canvas` y omita el método `paint()` utilizando `drawRect()` y `fillRect()` para dibujar el trazado de un rectángulo y rellenarlo.

```
class RectangleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width  = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawRect(width/2,height/2,70,30);
        g.fillRect(width/2,height/2,70,30);
    }
}
```

4. En la versión omitida del método `startApp()`, cree una instancia de la clase personalizada `RectangleCanvas`. Llame al método estático `setCurrent()` del objeto `Display` con el fin de configurar la clase `Displayable` actual según la instancia de `RectangleCanvas`.

```
public void startApp()
{
   Displayable disp = new RectangleCanvas();
   Display.getDisplay(this).setCurrent(disp);
}
```

Ejemplo de código: dibujar un rectángulo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DrawARectangle extends MIDlet
{
    public void startApp()
    {
        Displayable disp = new RectangleCanvas();
        Display.getDisplay(this).setCurrent(disp);
    }
    public void destroyApp(boolean flag)
    {
    }
    public void pauseApp()
    {
    }
}
class RectangleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width  = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawRect(width/2,height/2,70,30);
        g.fillRect(width/2,height/2,70,30);
    }
}
```

Dibujar un círculo

Para dibujar un círculo en la pantalla del dispositivo BlackBerry®, debe extender la clase `Canvas` y omitir su método `paint()`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Para crear el marco para el MIDlet, extienda la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Extienda la clase `Canvas` y omita el método `paint()` utilizando `drawArc()` y `fillArc()` para dibujar el trazado de un círculo y rellenarlo.

```
class CircleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width  = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawArc(width/2,height/2,50,50,0,360);
        g.fillArc(width/2,height/2,50,50,0,360);
    }
}
```

4. En la versión omitida del método `startApp()`, cree una instancia de la clase Displayable personalizada `CircleCanvas`. Llame al método estático `setCurrent()` del objeto `Display` con el fin de configurar la clase Displayable actual para que sea la instancia de `CircleCanvas`.

```
public void startApp()
{
   Displayable disp = new CircleCanvas();
   Display.getDisplay(this).setCurrent(disp);
}
```

Ejemplo de código: dibujar un círculo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
        Displayable disp = new CircleCanvas();
        Display.getDisplay(this).setCurrent(disp);
    }
    public void destroyApp(boolean flag)
    {
    }
    public void pauseApp()
    {
    }
}
class CircleCanvas extends Canvas
{
    public void paint(Graphics g)
    {
        int width  = getWidth();
        int height = getHeight();
        g.setColor(0xCC0999);
        g.drawArc(width/2,height/2,50,50,0,360);
        g.fillArc(width/2,height/2,50,50,0,360);
    }
}
```

Mostrar los controles

Las subclases de la clase `Item` representan los controles. Hay subclases integradas de la clase `Item` que representan controles comunes. La siguiente tabla enumera estos controles integrados junto con los controles de la interfaz de usuario que se utilizan para la representación.

Subclase Item	Controles de la interfaz de usuario representados
ChoiceGroup	Botón de opción, casilla de verificación y lista desplegable
DateField	Calendario y control de tiempo
Gauge	DateField
ImageItem	Control de imagen

Subclase Item	Controles de la interfaz de usuario representados
Spacer	Control de espacio
StringItem	Etiqueta
TextField	Control de casilla de edición

Además de estos controles integrados, hay una clase abstracta denominada `CustomItem` que le permite crear controles personalizados. Para crear un control personalizado, debe crear una clase que extienda `CustomItem` y omita los cinco métodos abstractos en dicha clase. Cuatro de los métodos abstractos se relacionan con la medición del control. El quinto método es `paint()`. Se encuentra en la versión omitida del método `paint()` en el cual ha escrito el código para dibujar su control personalizado.

Independientemente de si utiliza un control personalizado integrado, los pasos para mostrar el control son los mismos.

- Cree una instancia del control.
- Defina las propiedades del control.
- Cree una instancia de la clase `Form`.
- Utilice el método `append()` de la instancia `Form` para agregar el control al formulario.
- Utilice el método `setCurrent()` de la clase `Display` para mostrar el formulario que contiene el control.

Mostrar una etiqueta

Para mostrar una etiqueta en la pantalla de un dispositivo BlackBerry®, debe crear las instancias de un objeto `Form` y un objeto `StringItem`. Puede pasar el texto para que aparezca en el control como parámetros para la clase del constructor.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DisplayAnEditBox extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `StringItem`.

```
public class DisplayALabel extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private StringItem mLabel;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones de casilla de verificación y agréguela a un objeto `Form`.
   a. Cree una instancia de la clase `StringItem` para representar la etiqueta. Especifique `StringItem.PLAIN` en el constructor para indicar que la etiqueta debe aparecer como texto sin formato. El resto de opciones de apariencia disponibles provocan que la etiqueta aparezca como un botón o un hipervínculo.

```
public void startApp()
{
   mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar la etiqueta al formulario.

```
public void startApp()
{
   mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
   mForm = new Form("Display a Label");
   mForm.append(mLabel);
}
```

   c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
   mForm = new Form("Display a Label");
   mForm.append(mLabel);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar un ejemplo

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayALabel extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private StringItem mLabel;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mLabel = new StringItem("Label text","String text",StringItem.PLAIN);
        mForm = new Form("Display a Label");
        mForm.append(mLabel);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

Mostrar un campo de texto

Para mostrar un campo de texto en la pantalla de un dispositivo BlackBerry®, debe crear las instancias de un objeto `Form` y un objeto `TextField`. Debe incluir el objeto `TextField` en el objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

   ```
   import javax.microedition.midlet.*;
   import javax.microedition.lcdui.*;
   ```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

   ```
   public class DisplayATextField extends MIDlet
   {
       public void startApp()
       {
       }
       public void pauseApp()
       {
       }
   ```

```
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `TextField`. Cree, también, una constante denominada MAXCHARS. La constante se utilizará para especificar el número máximo de caracteres que se pueden introducir en el campo de texto.

```
public class DisplayATextField extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private TextField mTextField;
    private static final int MAXCHARS = 100;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `TextField` y agréguela a un objeto `Form`.
   a. Cree una instancia de la clase `TextField` para representar el campo de texto. Especifique `MAXCHARS` en el constructor para limitar el número de caracteres que pueden introducirse en el campo de texto.

```
public void startApp()
{
   mEditBox = new TextField("Text Field Label:",null,MAXCHARS,0);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar el campo de texto al formulario.

```
public void startApp()
{
   mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
   mForm = new Form("MIDlet Developer Guide: Display a text field.");
   mForm.append(mTextField);
}
```

   c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
   mForm = new Form("MIDlet Developer Guide: Display a text field.");
   mForm.append(mEditBox);
```

```
    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar un campo de texto

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayATextField extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private TextField mTextField;
    private static final int MAXCHARS = 100;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
        mForm = new Form("MIDlet Developer Guide: Display a text field.");
        mForm.append(mTextField);
        mDisplay.setCurrent(mForm);
    }
    public void destroyApp(boolean flag)
    {
    }
    public void pauseApp()
    {
    }
}
```

Mostrar botones de opción

Para mostrar los botones de opción en la pantalla de un dispositivo BlackBerry®, debe crear las instancias de un objeto `Form` y un objeto `ChoiceGroup`. Debe incluir las opciones de los botones de opción en el objeto `ChoiceGroup` y, a continuación, incluir el objeto `ChoiceGroup` en el objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase MIDlet y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
```

```
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `ChoiceGroup`.

```
public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    // MIDlet lifecycle method overrides omitted
}
```

4. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `ChoiceGroup`.

```
public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    // MIDlet lifecycle method overrides omitted
}
```

5. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de botones de opción. Especifique `Choice.EXCLUSIVE` en el constructor ChoiceGroup para indicar que el grupo de opciones se excluyen mutuamente y que debe ser presentado como botones de opción. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada botón de opción.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
}
```

6. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones del botón de selección y agréguela a un objeto `Form`.

a. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de botones de opción. Especifique `Choice.Exclusive` en el constructor `ChoiceGroup` para indicar que el grupo de opciones se excluyen mutuamente y que debe ser presentado como botones de opción. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada opción de botón de opción.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
}
```

b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar al formulario el conjunto de botones de opción almacenado en la variable `mChoices`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Display Option Buttons");
   mForm.append(mChoices);
}
```

c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Display Option Buttons");
   mForm.append(mChoices);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar botones de opción

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayOptionButtons extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm = new Form("Display Option Buttons");
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Display option
buttons",Choice.EXCLUSIVE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm.append(mChoices);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

Mostrar casillas de verificación

Para mostrar casillas de verificación en la pantalla de un dispositivo BlackBerry®, debe crear un objeto `Form` y un objeto `ChoiceGroup`. Debe incluir las opciones de casilla de verificación en el objeto `ChoiceGroup` y, a continuación, incluir el objeto `ChoiceGroup` en el objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Para crear el marco para el MIDlet, extienda la clase `MIDlet` y omita los métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
```

```
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `ChoiceGroup`.

```
public class DisplayCheckBoxes extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones de casilla de verificación y agréguela a un objeto `Form`.
   a. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de casillas de verificación. Especifique `Choice.MULTIPLE` en el constructor ChoiceGroup para indicar que puede seleccionar más de una de las opciones presentadas y que éstas deben, por lo tanto, ser presentadas como casillas de verificación. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada casilla de verificación.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar al formulario el conjunto de casillas de verificación almacenado en la variable `mChoices`.

```
public void startApp()
{
  mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
```

```
    mForm = new Form("Display Check Boxes");
    mForm.append(mChoices);
}
```

c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Display check
boxes",Choice.MULTIPLE);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
    mForm = new Form("Display Check Boxes");
    mForm.append(mChoices);
    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar casillas de verificación

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayCheckBoxes extends MIDlet
{
    Display mDisplay;
    Form mForm;
    ChoiceGroup mChoices;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Display
checkboxes",Choice.MULTIPLE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm = new Form("Display Check Boxes");
        mForm.append(mChoices);
        mDisplay.setCurrent(mForm);
    }
    public void destroyApp(boolean flag)
    {
    }
    public void pauseApp()
```

```
    {
    }
}
```

Comandos para la pantalla

Para crear una interfaz de usuario mediante la API de MIDP, debe recuperar en primer lugar una instancia de la clase `Display` que represente la pantalla en el dispositivo BlackBerry®. A continuación, debe crear una instancia de una clase que represente algo que se pueda mostrar en la pantalla de un dispositivo. Todas estas clases se derivan de la clase `Displayable`. Por último, debe utilizar el método `setCurrent()` de la clase `Display`, pasándole la instancia de la clase `Displayable`, para mostrar algo en la pantalla del dispositivo.

Todas las clases que se derivan de `Displayable` heredan la capacidad de tener comandos asociados a las mismas. Un comando es un elemento de interfaz de usuario que un dispositivo MIDP puede representar de muy distintas maneras. Puede mostrarse como un botón, un menú o de cualquier otra forma que permita al usuario del dispositivo interactuar con él. Al escribir código para crear un comando, no es posible controlar exactamente cómo aparecerá el comando o dónde aparecerá. Puede proporcionar tanto una etiqueta textual corta como larga para el comando y puede especificar el tipo de comando y asignarle una prioridad relativa. En el código que describe el comando, debe declarar lo que desea, cada dispositivo es libre de interpretar dichas declaraciones e implementarlas de forma que tengan el mayor sentido para dicho dispositivo en particular.

Los comandos se representan a través de la clase `Command`. Para mostrar un comando, debe crear una instancia nueva de la clase `Command` y asociarla con una instancia de la clase `Displayable`. Debe establecer la asociación pasando la instancia de la clase `Command` como un parámetro al método `addCommand()` de la clase `Displayable`.

1. Importe las dos bibliotecas de MIDP necesarias. La clase `MIDlet` está en el espacio de nombres `midlet` y las clases `Command` y `Form` están en el espacio de nombres `lcdui`.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DisplayCommands extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Agregue las variables de miembro privadas. Agregue las variables de miembro privadas a la clase de la aplicación MIDlet derivada para almacenar las referencias a las instancias de las clases `Display`, `Form` y `Command`.

```
public class DisplayCommands extends MIDlet
{
    private Display mDisplay;
    private Form    mForm;
    private Command mCommand;
    //MIDlet lifecycle methods omitted
}
```

4. Rellene las variables de miembro privadas. En la versión omitida del método `startApp()`, utilice el método estático `getDisplay` de la clase `Display` para recuperar una instancia de la clase correspondiente a la pantalla en el dispositivo actual. Rellene la variable `mDisplay` con la instancia devuelta. Cree las instancias nuevas de las clases `Form` y `Command` y almacene las referencias a las mismas en las variables de miembro `mForm` y `mCommand`. La cadena pasada al constructor de la clase `Form` aparecerá como un título en el formulario cuando se muestre en el dispositivo. Las primeras dos cadenas pasadas al constructor de la clase `Command` son etiquetas cortas y largas. El tercer parámetro indica el tipo de comando. En este caso, se especifica `Command.SCREEN` para indicar que el comando no es un comando estándar como Back o Help. El parámetro final configura la prioridad relativa del comando. Hay sólo un comando en este ejemplo, pero el valor 0 se especifica para garantizar que este comando sea tratado con prioridad relativa alta si otros comandos se agregan en el futuro.

```
public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mForm    = new Form("MIDlet Developers Guide: Display Commands");
    mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
}
```

5. Asocie el comando con la clase `Displayable`.

   La clase `Displayable` es un formulario en este ejemplo. El método `addCommand()` de la clase Form se utiliza para asociar la instancia del comando almacenada en `mCommand` al formulario al que se hace referencia a través de `mForm`.

```
public void startApp()
{
    mDisplay = Display.getDisplay(this);
    mForm    = new Form("MIDlet Developers Guide: Display Commands");
    mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
    mForm.addCommand(mCommand);
}
```

6. Configure la clase actual `Displayable` según la que tiene el comando asociado.

   El formulario al que hace referencia `mForm` tiene un comando asociado. Utilice el método `setCurrent()` de la clase `Display` con el fin de configurar la clase `Displayable` actual según `mForm`.

```
public void startApp()
{
   mDisplay = Display.getDisplay(this);
   mForm    = new Form("MIDlet Developers Guide: Display Commands");
   mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
   mForm.addCommand(mCommand);
   mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar comandos

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayCommands extends MIDlet
{
    Display mDisplay;
    Form    mForm;
    Command mCommand;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mForm    = new Form("MIDlet Developers Guide: Display Commands");
        mCommand = new Command("Short Label","A Long Command Label",Command.SCREEN,0);
        mForm.addCommand(mCommand);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean bForce)
    {
    }
}
```

Mostar un menú desplegable

Para mostrar un menú desplegable en la pantalla de un dispositivo BlackBerry®, debe crear las instancias de un objeto `Form` y un objeto `ChoiceGroup`. Una vez cree el objeto `ChoiceGroup`, debe especificar que es del tipo `Choice.POPUP`. Esta opción se representa en el dispositivo como un menú desplegable. Debe incluir las opciones del menú desplegable en el objeto `ChoiceGroup` y, a continuación, incluir el objeto `ChoiceGroup` en el objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase MIDlet y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class DrawACircle extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `ChoiceGroup`.

```
public class DisplayCheckBoxes extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones de casilla de verificación y agréguela a un objeto `Form`.
   a. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de casillas de verificación. Especifique `Choice.POPUP` en el constructor `ChoiceGroup` para indicar que el grupo de opciones es mutuamente exclusivo y que debe ser presentado bajo la forma de un menú desplegable. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada opción del menú desplegable.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar al formulario el menú desplegable almacenado en la variable `mChoices`.

```
public void startApp()
{
  mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Display a drop-down");
   mForm.append(mChoices);
}
```

c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada mDisplay. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Display a drop-down");
   mForm.append(mChoices);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

Ejemplo de código: mostrar un menú desplegable

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DisplayADropDown extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Display a drop-
down",Choice.POPUP);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm = new Form("Display a drop-down");
        mForm.append(mChoices);
```

```
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

Gestionar la entrada

Para recuperar y procesar la entrada en un dispositivo , debe implementar una interfaz de oyente en una clase utilizada por un MIDlet. A continuación, debe asociar el oyente con una clase UI derivada de `Displayable`. A la clase UI se le notifica la entrada y pasa información acerca de la misma a los métodos de oyente adecuados y registrados para su procesamiento.

Hay dos interfaces de oyente: `CommandListener` y `ItemStateListener`. Cada uno de ellos incluye un método único al cual se llama para procesar la entrada. El método único de la interfaz `CommandListener` se denomina `commandAction()` y acepta los valores `Command` y `Displayable` como parámetros. Si la entrada ha sido iniciada por un comando, el valor del parámetro `Command` identifica la instancia de la clase correspondiente `Command`. El valor del parámetro `Displayable` se corresponde con la instancia de la clase `Displayable` que representa el componente de la interfaz de usuario en el cual se recibió el evento.

Responder a comandos

Para responder a los comandos debe implementar la interfaz `CommandListener`. También debe asociar el oyente con el formulario que tiene los comandos asociados al mismo. Debe hacer la asociación utilizando el método `setCommandListener()` del objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`. Implemente la interfaz `CommandListener` incluyendo el método `commandAction()`.

```
public class RespondToCommands extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
```

```
    public void destroyApp(boolean flag)
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form` y `Command`.

```
public class RespondToCommands extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form    mForm;
    private Command mCommand1;
    private Command mCommand2;
```

4. En la versión omitida del método `startApp()`, cree instancias del objeto Command y agréguelas a un objeto Form.
   a. Cree dos instancias nuevas de la clase `Command` para representar dos comandos distintos que procesar.

```
public void startApp()
{
   mCommand1 = new Command("Short Label 1","A Long Command Label
1",Command.SCREEN,0);
   mCommand2 = new Command("Short Label 2","A Long Command Label
2",Command.SCREEN,0);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `addCommand()` del objeto `Form` para agregar los comandos al formulario. Por último, utilice el método `setCommandListener()` de la clase `Form` para asociar un oyente con el formulario. Al pasar esta palabra clave a `setCommandListener()` se especifica que la interfaz del oyente se implementa por la propia clase `MIDlet`. La implementación de una clase en la interfaz `commandAction()` se describe a continuación.

```
public void startApp()
{
   mCommand1 = new Command("Short Label 1","A Long Command Label
1",Command.SCREEN,0);
   mCommand2 = new Command("Short Label 2","A Long Command Label
2",Command.SCREEN,0);
   mForm    = new Form("MIDlet Developerment Guide: Display Commands");
   mForm.addCommand(mCommand1);
   mForm.addCommand(mCommand2);
   mForm.setCommandListener(this);
}
```

c. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mCommand1 = new Command("Short Label 1","A Long Command Label
1",Command.SCREEN,0);
   mCommand2 = new Command("Short Label 2","A Long Command Label
2",Command.SCREEN,0);
   mForm    = new Form("MIDlet Developers Guide: Display Commands");
   mForm.addCommand(mCommand1);
   mForm.addCommand(mCommand2);
   mForm.setCommandListener(this);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

5. Para determinar el comando invocado, utilice un constructor if/else para comparar la instancia de la clase `Command` pasada al método `commandAction()` con los comandos implementados. Muestre un mensaje indicando qué comando se ha invocado mediante el método `append()` de la clase `Form`.

```
public void commandAction(Command c, Displayable d)
    {
        if (c == mCommand1)
        {
            mForm.append("You invoked command 1.");
        }
        else if (c == mCommand2)
        {
            mForm.append("You invoked command 2.");
        }
    }
```

Ejemplo de código: responder a comandos

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToCommands extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form    mForm;
    private Command mCommand1;
    private Command mCommand2;
    public void startApp()
    {
```

```
        mDisplay = Display.getDisplay(this);
        mForm    = new Form("MIDlet Developers Guide: Display Commands");
       mCommand1 = new Command("Short Label 1","A Long Command Label 1",Command.SCREEN,
0);
       mCommand2 = new Command("Short Label 2","A Long Command Label 2",Command.SCREEN,
0);
        mForm.addCommand(mCommand1);
        mForm.addCommand(mCommand2);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean bForce)
    {
    }
    public void commandAction(Command c, Displayable d)
    {
        if (c == mCommand1)
        {
            mForm.append("You invoked command 1.");
        }
        else if (c == mCommand2)
        {
            mForm.append("You invoked command 2.");
        }
    }
}
```

Responder a la entrada de controles

Para responder a la entrada de controles debe extender las clase MIDlet e implementar la interfaz `CommandListener`. Debe asociar el oyente con el formulario que contiene los controles. El formulario reenvía la entrada de dichos controles a su implementación del método `commandAction()` de la interfaz `CommandListener`.

Responder a la información introducida en un campo de texto

Para responder a la información introducida en un campo de texto debe implementar la interfaz `CommandListener`. También debe asociar el oyente con el formulario que contiene la instancia de la clase `TextField` que representa la casilla de edición. Debe hacer la asociación utilizando el método `setCommandListener()` del objeto `Form`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`. Implemente la interfaz `CommandListener` incluyendo el método `commandAction()`.

```
public class RespondToTextField extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form`, `TextField` y `Command`. Cree, también, una constante denominada `MAXCHARS`. La constante se utilizará para especificar el número máximo de caracteres que se pueden introducir en la casilla de edición.

```
public class RespondToTextField extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private TextField mEditBox;
    private Command mSubmitCmd;
    private static final int MAXCHARS = 100;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto TextField y agréguela a un objeto Form.
   a. Cree una instancia de la clase `TextField` para representar la casilla de edición. Especifique MAXCHARS en el constructor para limitar el número de caracteres que pueden introducirse en la casilla de edición.

```
public void startApp()
{
   mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
}
```

   b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar la casilla de edición al formulario.

```
public void startApp()
{
   mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
   mForm = new Form("MIDlet Developer Guide: Respond to information entered in
an edit box.");
   mForm.append(mEditBox);
}
```

c. Cree una instancia nueva de la clase `Command` y almacénela en la variable privada `mSubmitCmd`. Especifique `Command.OK` como segundo parámetro en el constructor para indicar al marco MIDP que el comando debe comportarse como un botón Aceptar o Enviar. Utilice el método `addCommand()` de la clase `Form` para asociar el comando con `mForm`, la instancia de la clase `Form` que contiene el campo de texto. Por último, utilice el método `setCommandListener()` de la clase `Form` para asociar un oyente con el formulario. Al pasar `esta` palabra clave a `setCommandListener()` se especifica que la interfaz del oyente se implementa por la propia clase `MIDlet`. La implementación de una clase en la interfaz `commandAction()` se describe a continuación.

```
public void startApp()
{
        mEditBox = new TextField("Text Field Label:",null,MAXCHARS,0);
       mForm = new Form("MIDlet Developer Guide: Respond to information entered
in a text field.");
        mForm.append(mEditBox);
        mSubmitCmd = new Command("","Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
}
```

d. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
 public void startApp()
 {
        mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
       mForm = new Form("MIDlet Developer Guide: Respond to information entered
in an edit box.");
        mForm.append(mEditBox);
        mSubmitCmd = new Command("","Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay = Display.getDisplay(this);
        mDisplay.setCurrent(mForm);
 }
```

5. Para recuperar el texto introducido en el campo de texto, utilice el método `getString()` de la instancia `TextField` almacenada en `mTextField`. Muéstrela en el formulario utilizando el método `append()` de la clase `Form`.

```
public void commandAction(Command c, Displayable disp)
{
   String strEnteredText = mTextField.getString();
   mForm.append("You entered " + strEnteredText);
}
```

Responder a la selección de los botones de opción

Los botones de opción se representan con una instancia de la clase `ChoiceGroup` con la opción `Choice.EXCLUSIVE` especificada. El seleccionar un botón de selección en particular no genera como resultado automáticamente una acción. Debe asociar y registrar un objeto listener para responder a una selección enviada.

Puede crear un `ItemStateListener` y registrarlo para controlar cualquier cambio en el estado de los elementos en un formulario. En el objeto listener, podría actuar basándose en la selección realizada por un usuario. Este enfoque no permite al usuario cambiar la selección inicial. Para ofrecer al usuario dicha opción, cree un comando submit independiente y un `CommandListener` correspondiente en lugar de un `ItemStateListener`. Utilizando este método, los usuarios pueden cambiar la selección de los botones de selección hasta estar satisfechos con su elección. A continuación, pueden utilizar el comando enviar para proporcionar su elección final. Debe responder a dicha selección final en el método commandAction() de la interfaz CommandListener.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`. Implemente la interfaz `CommandListener` incluyendo el método `commandAction()`.

```
public class RespondToOptionButtonSelection extends MIDlet implements
CommandListener
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de las clases `Display`, Form, `ChoiceGroup` y `Command`.

```
public class RespondToOptionButtonSelection extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones del botón de selección y agréguela a un objeto `Form`.

   a. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de botones de opción. Especifique `Choice.Exclusive` en el constructor `ChoiceGroup` para indicar que el grupo de opciones es mutuamente exclusivo y que debe ser presentado como botones de selección. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada opción de botón de opción.

   ```
   public void startApp()
   {
      mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to radio button
   selection",Choice.EXCLUSIVE);
      mChoices.append("First Choice", null);
      mChoices.append("Second Choice", null);
      mChoices.append("Third Choice", null);
   }
   ```

   b. Cree una instancia nueva de la clase Form. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método append() del objeto Form para agregar al formulario el conjunto de botones de selección almacenado en la variable mChoices.

   ```
   public void startApp()
   {
      mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
                     "Respond to radio button selection",Choice.EXCLUSIVE);
      mChoices.append("First Choice", null);
      mChoices.append("Second Choice", null);
      mChoices.append("Third Choice", null);
      mForm = new Form("Respond to Radio Button Selection");
      mForm.append(mChoices);
   }
   ```

   c. Cree una instancia nueva de la clase `Command` y almacénela en la variable privada `mSubmitCmd`. Especifique `Command.OK` como segundo parámetro en el constructor para indicar al marco MIDP que el comando debe comportarse como un botón Aceptar o Enviar. Utilice el método `addCommand()` de la clase Form para asociar el comando con mForm, la instancia de la clase `Form` que contiene las casillas de verificación. Por último, utilice el

método `setCommandListener()` de la clase Form para asociar un oyente con el formulario. Al pasar esta palabra clave a `setCommandListener()` se especifica que la interfaz del oyente se implementa por la propia clase `MIDlet`. La implementación de una clase en la interfaz `commandAction()` se describe a continuación.

```
public void startApp()
{
       mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm = new Form("Respond to Checkbox Selection");
        mForm.append(mChoices);
        mSubmitCmd = new Command("Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
}
```

d. Utilice el método estático `getDisplay()` de la clase Display para recuperar un objeto Display que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto Display para configurar el objeto Displayable actual en el formulario almacenado en la variable mForm.

```
public void startApp()
{
  mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
                  "Respond to radio button selection",Choice.EXCLUSIVE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Radio Button Selection");
   mForm.append(mChoices);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

Responder a la selección de la casilla de verificación

Para responder a la selección de la casilla de verificación en el dispositivo BlackBerry, debe implementar la interfaz `CommandListener`. También debe asociar el oyente con el formulario que contiene la instancia de la clase `ChoiceGroup` que representa las casillas de verificación. Debe hacer la asociación utilizando el método `setCommandListener()` del objeto Form. Por último, debe proporcionar una implementación del método único, `commandAction()`, en la interfaz `CommandListener`. En dicha implementación, debe determinar qué casillas de verificación ha seleccionado el usuario y poner en marcha la acción apropiada.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omitiendo los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`. Implemente la interfaz `CommandListener` incluyendo el método `commandAction()`.

```
public class RespondToCheckBoxSelection extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form`, `ChoiceGroup` y `Command`.

```
public class RespondToCheckBoxSelection extends MIDlet
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia del objeto `ChoiceGroup`, rellénela con las opciones de casilla de verificación y agréguela a un objeto `Form`.
   a. Cree una instancia de la clase `ChoiceGroup` para representar el conjunto de casillas de verificación. Especifique `Choice.MULTIPLE` en el constructor `ChoiceGroup` para indicar que el grupo de opciones es mutuamente exclusivo y que debe ser presentado como casillas de verificación. Utilice el método `append()` de la instancia `ChoiceGroup` para crear y agregar una etiqueta a cada casilla de verificación.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
   mChoices.append("First Choice", null);
```

```
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
}
```

b. Cree una instancia nueva de la clase `Form`. El constructor selecciona un parámetro de valor String para mostrarlo como el título del formulario. Utilice el método `append()` del objeto `Form` para agregar al formulario el conjunto de casillas de verificación almacenado en la variable `mChoices`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Checkbox Selection");
   mForm.append(mChoices);
}
```

c. Cree una instancia nueva de la clase `Command` y almacénela en la variable privada `mSubmitCmd`. Especifique `Command.OK` como segundo parámetro en el constructor para indicar al marco MIDP que el comando debe comportarse como un botón Aceptar o Enviar. Utilice el método `addCommand()` de la clase `Form` para asociar el comando con `mForm`, la instancia de la clase `Form` que contiene las casillas de verificación. Por último, utilice el método `setCommandListener()` de la clase `Form` para asociar un oyente con el formulario. Al pasar esta palabra clave a `setCommandListener()` se especifica que la interfaz del oyente se implementa por la propia clase `MIDlet`. La implementación de una clase en la interfaz `commandAction()` se describe a continuación.

```
public void startApp()
{
       mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm = new Form("Respond to Checkbox Selection");
        mForm.append(mChoices);
        mSubmitCmd = new Command("Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
}
```

d. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` para configurar el objeto Displayable actual en el formulario almacenado en la variable `mForm`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
```

```
Selection",Choice.MULTIPLE);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Checkbox Selection");
   mForm.append(mChoices);
   mSubmitCmd = new Command("Submit",Command.OK,0);
   mForm.addCommand(mSubmitCmd);
   mForm.setCommandListener(this);
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mForm);
}
```

5. Cree una matriz booleana, `arrSelected[]`, con el mismo número de elementos que de casillas de verificación. Llame al método `getSelectedFlags()` del objeto `ChoiceGroup` para rellenar la matriz booleana. Después de la llamada al método, la matriz contendrá los valores del elemento true en las posiciones de matriz que se corresponden con los valores cero de las casillas de verificación seleccionadas y del elemento en las posiciones de matriz correspondientes a las casillas de verificación no seleccionadas.

```
public void commandAction(Command c, Displayable disp)
{
        if(c == mSubmitCmd)
        {
           boolean arrSelected[] = new boolean[mChoices.size()];
           mChoices.getSelectedFlags(arrSelected);
        }
}
```

6. Para mostrar un mensaje que incluya los nombres de cada una de las selecciones de las casillas de verificación junto con el hecho de si se han seleccionado o no antes del envío, utilice un bucle for para iterar a través de la matriz de banderas booleanas. Utilice el método `getString()` del objeto `ChoiceGroup` para recuperar sucesivamente cada una de las etiquetas de las casillas de verificación. Compruebe la matriz booleana en el índice actual, i, para determinar si se ha seleccionado o no la visualización. Una vez se ha construido la cadena del mensaje, muéstrela en el formulario mediante el método `append()` de la clase `Form`.

```
public void commandAction(Command c, Displayable disp)
{
        if(c == mSubmitCmd)
        {
           boolean arrSelected[] = new boolean[mChoices.size()];
           mChoices.getSelectedFlags(arrSelected);
           for(int i = 0; i < mChoices.size(); i++)
           {
              String strMessage;
              strMessage = mChoices.getString(i);
              if(arrSelected[i])
              {
                  strMessage += " was selected.";
```

```
                }
                else
                {
                    strMessage += " was not selected.";
                }
                mForm.append(strMessage);
            }
        }
}
```

Responder a una selección del menú desplegable

1. Importe las siguientes bibliotecas de aplicaciones MIDlet:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase `MIDlet` y omita los tres métodos obligatorios del ciclo de vida del MIDlet: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class RespondToDropDownSelection extends MIDlet implements CommandListener
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Implemente el método `commandAction()` de la interfaz `CommandListener` para responder a la selección desplegable.

```
public class RespondToDropDownSelection extends MIDlet implements CommandListener
{
    // MIDlet lifecycle method overrides omitted
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

4. Cree las variables privadas para almacenar instancias de las clases `Display`, `Form`, `ChoiceGroup` y `Command`.

```
public class RespondToDropDownSelection extends MIDlet
{
```

```
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    // MIDlet lifecycle method overrides omitted
}
```

5. Cree el menú desplegable.
   a. Utilice la palabra clave `new` para crear una instancia de la clase `ChoiceGroup`. En el constructor, proporcione un título para el menú desplegable y especifique `Choice.POPUP` para presentar el grupo de opciones como un menú desplegable.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
}
```

   b. Utilice el método `append()` de la clase `ChoiceGroup` para agregar elementos al menú desplegable.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
   mChoices.append("First Choice");
   mChoices.append("Second Choice");
   mChoices.append("Third Choice");
}
```

6. Agregue el menú desplegable a un formulario.
   a. Utilice la palabra clave `new` para crear una instancia de la clase `Form`. En el constructor, proporcione un título para el formulario.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-
Down",Choice.POPUP);
   mChoices.append("First Choice");
   mChoices.append("Second Choice");
   mChoices.append("Third Choice");
   mForm = new Form("Respond to Drop-Down Selection");
}
```

   b. Utilice el método `append()` de la clase `Form` para agregar el menú desplegable al formulario.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
```

```
Selection",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Drop-Down Selection");
   mForm.append(mChoices);
}
```

7. Configure un comando submit.
   a. Utilice la palabra clave `new` para crear una instancia de la clase `Command`. En el constructor, proporcione un subtítulo para el comando, el tipo de comando y su prioridad.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Drop-Down Selection");
   mForm.append(mChoices);
   mSubmitCmd = new Command("Submit",Command.OK,0);
}
```

   b. Utilice el método `addCommand()` de la clase `Form` para asociar el comando al formulario.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
   mForm = new Form("Respond to Drop-Down Selection");
   mForm.append(mChoices);
   mSubmitCmd = new Command("Submit",Command.OK,0);
   mForm.addCommand(mSubmitCmd);
}
```

   c. Pase la palabra clave `this` al método `setCommandListener()` de la clase `Form` para especificar que el oyente está implementado en la clase `RespondToDropDownSelection`.

```
public void startApp()
{
   mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
   mChoices.append("First Choice", null);
   mChoices.append("Second Choice", null);
   mChoices.append("Third Choice", null);
```

```
    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);
    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
}
```

8. Muestre el formulario que contiene el menú desplegable.
   a. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar una instancia de la clase `Display` que represente la visualización actual.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);
    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
    mDisplay = Display.getDisplay(this);
}
```

   b. Pase la variable local `mForm` al método `setCurrent()` de la clase `Display` para mostrar el formulario en la pantalla.

```
public void startApp()
{
    mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Drop-Down
Selection",Choice.POPUP);
    mChoices.append("First Choice", null);
    mChoices.append("Second Choice", null);
    mChoices.append("Third Choice", null);
    mForm = new Form("Respond to Drop-Down Selection");
    mForm.append(mChoices);
    mSubmitCmd = new Command("Submit",Command.OK,0);
    mForm.addCommand(mSubmitCmd);
    mForm.setCommandListener(this);
    mDisplay = Display.getDisplay(this);
    mDisplay.setCurrent(mForm);
}
```

9. Escriba código para responder a la selección de un elemento de la lista desplegable.
   a. Implemente el método `commandAction()` de la interfaz `CommandListener`.

```
public class RespondToDropDownSelection extends MIDlet implements
CommandListener
{
    // MIDlet lifecycle method overrides omitted
    public void commandAction(Command c, Displayable disp)
    {
    }
}
```

b. Utilice una declaración `if` para filtrar el comando submit.

```
public class RespondToDropDownSelection extends MIDlet implements
CommandListener
{
    // MIDlet lifecycle method overrides omitted
    public void commandAction(Command c, Displayable disp)
    {
       if(c == mSubmitCmd)
       {
       }
    }
}
```

c. Utilice el método `getSelectedIndex()` para recuperar el índice del elemento desplegable seleccionado.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
       int index = mChoices.getSelectedIndex();
    }
}
```

d. Pase el índice al método `getString()` para recuperar el texto asociado al elemento desplegable seleccionado.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
       int index = mChoices.getSelectedIndex();
       String strChoiceText = mChoices.getString(index);
    }
}
```

e. Utilice el método `append()` de la clase `Form` para mostrar un mensaje en la pantalla que indique qué elemento desplegable se ha seleccionado.

```
public void commandAction(Command c, Displayable disp)
{
    if(c == mSubmitCmd)
    {
       int index = mChoices.getSelectedIndex();
       String strChoiceText = mChoices.getString(index);
       mForm.append("You selected " + strChoiceText);
    }
}
```

Responder a la entrada del teclado

Para responder a la entrada del teclado, debe omitir el método `keyPressed()` de la clase `Canvas`.

1. Importe las dos bibliotecas de aplicaciones MIDlet necesarias.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

2. Cree el marco para el MIDlet extendiendo la clase MIDlet y omita los tres métodos obligatorios: `startApp()`, `pauseApp()` y `destroyApp()`.

```
public class RespondToKeyboard extends MIDlet
{
    public void startApp()
    {
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
}
```

3. Cree las variables privadas para almacenar instancias de los objetos `Display`, `Form`, `ChoiceGroup` y `Command`.

```
public class RespondToKeyboard extends MIDlet
{
   private Display mDisplay;
   private KeyCanvas mKeyCanvas;
    // MIDlet lifecycle method overrides omitted
}
```

4. En la versión omitida del método `startApp()`, cree una instancia de la clase personalizada `KeyCanvas()`. Utilice el método estático `getDisplay()` de la clase `Display` para recuperar un objeto `Display` que represente la visualización actual. Almacénelo en la variable privada `mDisplay`. Utilice el método `setCurrent()` del objeto `Display` con el fin de configurar el objeto Displayable actual en la clase personalizada Canvas.

```
public void startApp ()
{
   mKeyCanvas = new KeyCanvas();
   mDisplay = Display.getDisplay(this);
   mDisplay.setCurrent(mKeyCanvas);
}
```

5. Defina una extensión de la clase `Canvas`. Debe proporcionar una implementación del método `paint()`, pero en este caso, éste no tiene que hacer nada. Debe omitir el método `keyPressed()` para responder a la entrada del teclado. El ejemplo utiliza `System.out.println()` para mostrar un mensaje en la ventana de resultados cuando se pulsa una tecla con un código de tecla positivo.

```
class KeyCanvas extends Canvas
{
   public void paint(Graphics g)
   {
   }
   protected void keyPressed(int keyCode)
   {
      if (keyCode > 0)
      {
         System.out.println("You pressed " + ((char)keyCode));
      }
    }
};
```

Ejemplo de código: responder a la información introducida en un campo de texto

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToTextField extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private TextField mTextField;
    private Command mCommand;
    private static final int MAXCHARS = 100;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mTextField = new TextField("Text Field Label:",null,MAXCHARS,0);
```

```
        mForm = new Form("MIDlet Developer Guide: Respond to information entered in an
edit box.");
        mForm.append(mTextField);
        mCommand = new Command("","Submit",Command.OK,0);
        mForm.addCommand(mCommand);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
           String strEnteredText = mTextField.getString();
           mForm.append("You entered " + strEnteredText);
    }
}
```

Ejemplo de código: responder a la selección del botón de selección

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToRadioButtonSelection extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide:" +
                   "Respond to radio button selection",Choice.EXCLUSIVE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mSubmitCmd = new Command("Submit",Command.OK,0);
        mForm = new Form("Respond to Radio Button Selection");
        mForm.append(mChoices);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
    public void destroyApp(boolean flag)
    {
    }
```

```
    public void pauseApp()
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
        if(c == mSubmitCmd)
        {
           int index = mChoices.getSelectedIndex();
           String strChoiceText = mChoices.getString(index);
           mForm.append("You selected " + strChoiceText);
        }
    }
}
```

Ejemplo de código: responder a la selección de la casilla de verificación

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToCheckBoxSelection extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to Checkbox
Selection",Choice.MULTIPLE);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mSubmitCmd = new Command("Submit",Command.OK,0);
        mForm = new Form("Respond to Checkbox Selection");
        mForm.append(mChoices);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
    public void destroyApp(boolean flag)
    {
    }
    public void pauseApp()
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
        if(c == mSubmitCmd)
        {
```

```
            boolean arrSelected[] = new boolean[mChoices.size()];
            mChoices.getSelectedFlags(arrSelected);
            for(int i = 0; i < mChoices.size(); i++)
            {
               String strMessage;
               strMessage = mChoices.getString(i);
               if(arrSelected[i])
               {
                   strMessage += " was selected.";
               }
               else
               {
                   strMessage += " was not selected.";
               }
               mForm.append(strMessage);
            }
        }
    }
}
```

Ejemplo de código: responder a una selección del menú desplegable

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToDropDown extends MIDlet implements CommandListener
{
    private Display mDisplay;
    private Form mForm;
    private ChoiceGroup mChoices;
    private Command mSubmitCmd;
    public void startApp()
    {
        mDisplay = Display.getDisplay(this);
        mChoices = new ChoiceGroup("MIDlet Developer Guide: Respond to drop-
down",Choice.POPUP);
        mChoices.append("First Choice", null);
        mChoices.append("Second Choice", null);
        mChoices.append("Third Choice", null);
        mForm = new Form("Display a Drop-down");
        mForm.append(mChoices);
        mSubmitCmd = new Command("","Submit",Command.OK,0);
        mForm.addCommand(mSubmitCmd);
        mForm.setCommandListener(this);
        mDisplay.setCurrent(mForm);
    }
    public void pauseApp()
    {
    }
    public void destroyApp(boolean flag)
```

```
    {
    }
    public void commandAction(Command c, Displayable disp)
    {
        if(c == mSubmitCmd)
        {
           boolean arrSelected[] = new boolean[mChoices.size()];
           mChoices.getSelectedFlags(arrSelected);
           for(int i = 0; i < mChoices.size(); i++)
           {
              String strMessage;
              strMessage = mChoices.getString(i);
              if(arrSelected[i])
              {
                  strMessage += " was selected.";
              }
              else
              {
                  strMessage += " was not selected.";
              }
              mForm.append(strMessage);
           }
        }
    }
}
```

Ejemplo de código: responder a una entrada del teclado

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class RespondToKeyboard extends MIDlet
{
   private Display mDisplay;
   public void startApp ()
   {
      mDisplay = Display.getDisplay(this);
      KeyCanvas mKeyCanvas = new KeyCanvas();
      mDisplay.setCurrent(mKeyCanvas);
   }
   public void pauseApp()
   {
   }
   public void destroyApp(boolean flag)
   {
   }
};
class KeyCanvas extends Canvas
{
   public void paint(Graphics g)
```

```
    {
    }
    protected void keyPressed(int keyCode)
    {
       if (keyCode > 0)
       {
          System.out.println("You pressed " + ((char)keyCode));
       }
     }
};
```

Glosario

6

API
Application Programming Interface (Interfaz de programación de aplicaciones)

JVM
Java® Virtual Machine

Java ME
Plataforma Java®, Micro Edition

JSR
Java® Specification Request

MIDP
Mobile Information Device Profile (Perfil de dispositivo móvil de información)

Comentarios

7

Para ofrecer comentarios acerca de este documento, visite www.blackberry.com/docsfeedback.

Historial de revisiones del documento

8

Fecha	Descripción
8 de febrero de 2010	Se ha actualizado el siguiente tema para mencionar que los dispositivos que ejecutan BlackBerry® Device Software versión 5.0 o posterior son compatibles con MIDP 2.1. • API de Java ME y de Java para BlackBerry

Aviso legal

9

DE TERCEROS AQUÍ MENCIONADOS INCLUIDOS SIN NINGÚN TIPO DE LIMITACIÓN CUALQUIERA DE LOS SIGUIENTES DAÑOS: DIRECTOS, RESULTANTES, EJEMPLARES, INCIDENTALES, INDIRECTOS, ESPECIALES, PUNITIVOS O AGRAVADOS, DAÑOS POR PÉRDIDA DE BENEFICIOS O INGRESOS, IMPOSIBILIDAD DE CONSEGUIR LOS AHORROS ESPERADOS, INTERRUPCIÓN DE LA ACTIVIDAD COMERCIAL, PÉRDIDA DE INFORMACIÓN COMERCIAL, PÉRDIDA DE LA OPORTUNIDAD DE NEGOCIO O CORRUPCIÓN O PÉRDIDA DE DATOS, IMPOSIBILIDAD DE TRANSMITIR O RECIBIR CUALQUIER DATO, PROBLEMAS ASOCIADOS CON CUALQUIER APLICACIÓN QUE SE UTILICE JUNTO CON PRODUCTOS Y SERVICIOS DE RIM, COSTES DEBIDOS AL TIEMPO DE INACTIVIDAD, PÉRDIDA DE USO DE LOS PRODUCTOS Y SERVICIOS DE RIM O PARTE DE ÉL O DE CUALQUIER SERVICIO DE USO, COSTE DE SERVICIOS SUSTITUTIVOS, COSTES DE COBERTURA, INSTALACIONES O SERVICIOS, COSTE DEL CAPITAL O CUALQUIER OTRA PÉRDIDA MONETARIA SIMILAR, TANTO SI DICHOS DAÑOS SE HAN PREVISTO O NO, Y AUNQUE SE HAYA AVISADO A RIM DE LA POSIBILIDAD DE DICHOS DAÑOS.

EN LA MEDIDA MÁXIMA EN QUE LO PERMITA LA LEY DE SU JURISDICCIÓN, RIM NO TENDRÁ NINGÚN OTRO TIPO DE OBLIGACIÓN O RESPONSABILIDAD CONTRACTUAL, EXTRACONTRACTUAL O CUALQUIER OTRA, INCLUIDA CUALQUIER RESPONSABILIDAD POR NEGLIGENCIA O RESPONSABILIDAD ESTRICTA.

LAS LIMITACIONES, EXCLUSIONES Y DESCARGOS DE RESPONSABILIDAD SE APLICARÁN: (A) INDEPENDIENTEMENTE DE LA NATURALEZA DE LA CAUSA DE LA ACCIÓN, DEMANDA O ACCIÓN SUYA, INCLUIDA PERO NO LIMITADA AL INCUMPLIMIENTO DEL CONTRATO, NEGLIGENCIA, AGRAVIO, EXTRACONTRACTUAL, RESPONSABILIDAD ESTRICTA O CUALQUIER OTRA TEORÍA DEL DERECHO Y DEBERÁN SOBREVIVIR A UNO O MÁS INCUMPLIMIENTOS ESENCIALES O AL INCUMPLIMIENTO DEL PROPÓSITO ESENCIAL DE ESTE CONTRATO O CUALQUIER SOLUCIÓN CONTENIDA AQUÍ; Y (B) A RIM Y A SUS EMPRESAS AFILIADAS, SUS SUCESORES, CESIONARIOS, AGENTES, PROVEEDORES (INCLUIDOS LOS PROVEEDORES DE SERVICIOS DE USO), DISTRIBUIDORES AUTORIZADOS POR RIM (INCLUIDOS TAMBIÉN LOS PROVEEDORES DE SERVICIOS DE USO) Y SUS RESPECTIVOS DIRECTORES, EMPLEADOS Y CONTRATISTAS INDEPENDIENTES.

ADEMÁS DE LAS LIMITACIONES Y EXCLUSIONES MENCIONADAS ANTERIORMENTE, EN NINGÚN CASO NINGÚN DIRECTOR, EMPLEADO, AGENTE, DISTRIBUIDOR, PROVEEDOR, CONTRATISTA INDEPENDIENTE DE RIM O CUALQUIER AFILIADO DE RIM ASUMIRÁ NINGUNA RESPONSABILIDAD DERIVADA DE O RELACIONADA CON LA DOCUMENTACIÓN.

Antes de instalar, usar o suscribirse a cualquiera de los Productos y servicios de terceros, es su responsabilidad asegurarse de que su proveedor de servicios de uso ofrezca compatibilidad con todas sus funciones. Puede que algunos proveedores de servicios de uso no ofrezcan las funciones de exploración de Internet con una suscripción al servicio BlackBerry® Internet Service. Consulte con su proveedor de servicios acerca de la disponibilidad, arreglos de itinerancia, planes de servicio y funciones. La instalación o el uso de los Productos y servicios de terceros con productos y servicios de RIM puede precisar la obtención de una o más patentes, marcas comerciales, derechos de autor u otras licencias para evitar que se vulneren o violen derechos de terceros. Usted es el único responsable de determinar si desea utilizar Productos y servicios de terceros y si se necesita para ello cualquier otra licencia de terceros. En caso de necesitarlas, usted es el único responsable de su adquisición. No instale o utilice Productos y servicios de terceros hasta que se hayan adquirido todas las licencias necesarias. Cualquier tipo de Productos y servicios de terceros que se proporcione con los productos y servicios de RIM se le facilita para su comodidad "TAL CUAL" sin ninguna condición expresa e implícita, aprobación, garantía de cualquier tipo por RIM y RIM no sume ninguna responsabilidad en relación con ello. El uso de los Productos y servicios de terceros se regirá y estará sujeto a la aceptación de los términos de licencias independientes aplicables en este caso con terceros, excepto en los casos cubiertos expresamente por una licencia u otro acuerdo con RIM.

Algunas funciones mencionadas en esta documentación requieren una versión mínima del software de BlackBerry® Enterprise Server, BlackBerry® Desktop Software y/o BlackBerry® Device Software.

Los términos de uso de cualquier producto o servicio de RIM se presentan en una licencia independiente o en otro acuerdo con RIM que se aplica en este caso. NINGUNA PARTE DE LA PRESENTE DOCUMENTACIÓN ESTÁ PENSADA PARA PREVALECER SOBRE CUALQUIER ACUERDO EXPRESO POR ESCRITO O GARANTÍA PROPORCIONADA POR RIM PARA PARTES DE CUALQUIER PRODUCTO O SERVICIO DE RIM QUE NO SEA ESTA DOCUMENTACIÓN.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canadá

Research In Motion UK Limited
Centrum House
36 Station Road
Egham, Surrey TW20 9LF
Reino Unido

Publicado en Canadá